RTS: Regression Test Suite

Amy Langenhorst

This is a howto document for the RTS (Regression Test Suite) for FMS. The RTS is a tool to facilitate running FMS models. The user creates a model description file in xml format (or uses a preexisting file) and uses various rts-utilities on it. The rts-utilities, written in perl, can acquire code, create and submit compile scripts, and create and submit run-scripts, among other things. A pdf version of this document is available at http://www.gfdl.noaa.gov/~arl/rts/rts.pdf.

HTML pages generated from DocBook are best viewed in Netscape 7.

Table of Contents

1. What is the RTS?

The RTS consists of:

- An XML file which contains all experiment-specific variables such as the cvs commands used to check out the code, the path of the initial conditions file, namelists, and runtime specifications. This is the only file which users need to edit in order to use the RTS.
- Two c-shell template scripts for compiling and running models. The user never needs to look at them to run the RTS.
- Four perl scripts which read the XML file and perform a specific function:
    - rtsmake: checks out the model's code if necessary. Creates and optionally submits compile scripts using a c-shell compile template.
    - rtsrun: creates and optionally submits runscripts based on a c-shell runscript template.
    - rtscheck: runs reproducibility tests on output from RTS runs. Also calculates timing and performance statistics (work in progress).
    - rtslist: lists the experiments in your xml file, optionally with descriptive information about each model as provided in the xml.
    - rtsstatus: reports on the status of your batch compiles and batch runs. This information is parsed from the batch stdout files.
- Several conventions upon which defaults are based, such as the directory structure and naming conventions.

2. What does the RTS test?

Balaji's document [http://www.gfdl.noaa.gov/~vb/rts/] describes the goals, policies, and technical details of the RTS.

3. Quickstart Guide

1. Execute the following cvs checkout:

   ```
   setenv CVSROOT /home/fms/cvs
   cvs co rts
   ```

   This will give you a directory called `rts/` and a file inside called `rts.xml`, which contains the xml for several experiments from the FMS Model Development Database.

   Warning

   If your RTS directory is something other than `/home/$USER/rts`, you will need to edit the fourth line in `rts.xml` accordingly:

   ```
      <directory type="root">$HOME/rts</directory>
   ```

   If you wish to put your archived output in a directory other than /archive/$USER/rts, you should edit the sixth line accordingly:

   ```
      <directory type="archive">/archive/$USER/rts</directory>
   ```

2. Change to the directory containing `rts.xml` and run **/home/fms/bin/rtslist -v** to view available experiments. See Appendix D for usage information on **rtslist**.

   - You must either run the rts-utilities from the directory containing `rts.xml`, or else give them an **-x** argument with the path to your XML file.
   - You can add `/home/fms/bin` to your Unix `$PATH` to make this (and the following commands) quicker to type.

3. Run **/home/fms/bin/rtsmake** only on the experiment(s) for which you want to check-out cvs code and/or compile. See Appendix A for usage information on **rtsmake**.

   - The cvs source will be placed in `$root/$name/src/`.
   - The compilation will be done in `$root/$name/exec/` and the executable will be created as `$root/$name/exec/fms_$name.x`.
   - The script's stdout will be placed in `$root/$name/exec/stdout`. You can monitor the progress of any RTS compile by monitoring the stdout file.
   - Use the **-s** option to automatically qsub the compile script to the AC. For example, if you want to check-out and compile am2p10 and mom4_test1, use: **/home/fms/bin/rtsmake -s am2p10 mom4_test1**.

4. Run **/home/fms/bin/rtsrun** on the experiment(s) you want to run. See Appendix B for usage information on **rtsrun**.

   - Again use the **-s** option to automatically qsub the scripts to the LSC.
   - To run all currently available tests (basic, scaling, restarts) on am2p10 and mom4_test1, use: **/home/fms/bin/rtsrun -s -r suite am2p10 mom4_test1**.
   - The scripts' stdout files will be placed in `/archive/$USER/rts/$name/$RUNPARAMS/ascii/stdout`. For details on the naming convention of $RUNPARAMS, see Appendix F.

5. Run **rtsstatus** to check the progress of the runs you've submitted as batch jobs. See Appendix E for usage information on **rtsstatus**. To see how far your compiles or runs have gotten for am2p10 and mom4_test1, use **/home/fms/bin/rtsstatus am2p10 mom4_test1**. It is helpful to use the command **qa -n -u $USER** along with **rtsstatus** to see which jobs are running or waiting in the queue.

6. When at least two of the runs have completed (successfully), you can use **rtscheck** to verify the reproducibility

over pe-counts and restarts. See Appendix C for usage information on **rtscheck**. To verify that the restart files for am2p10 and mom4_test1 have reproduced bit-for-bit, use: **/home/fms/bin/rtscheck am2p10 mom4_test1**.

4. Editing the XML

XML (Extensible Markup Language [http://www.xml.com/pub/a/98/10/guide0.html]) is a markup language similar to HTML, but where we've defined tags appropriate for our use in the RTS. The xml file is called `rts.xml` by default. This is a text file which you can edit with your favorite text editor. Vi (VIM), emacs, and nedit perform syntax highlighting automatically.

A sample xml file documents all the tags which are available. It is available syntax-highlighted in HTML format at http://www.gfdl.noaa.gov/~arl/rts_example.

A tool called 'pollo' is available for browsing xml at `/home/arl/bin/pollo`. Pollo provides an interactive, graphical view of any xml file. Currently pollo can mess up your spacing if you save from it, so I recommend it only as an xml browser until a newer version is released. To use pollo to view a file, execute **/home/arl/bin/pollo rts.xml**.

5. Inheritance

It is possible for an experiment to inherit parameters from another experiment in the same xml file. Sample XML illustrating inheritance is shown in http://www.gfdl.noaa.gov/~arl/rts_example. Aside from a few special cases, the rules governing inheritance are as follows.

- **rtsmake** and **rtsrun** will look inside the experiment for the data, such as `<gridSpec>`, for example.
- If the data is not found, **rtsmake** and **rtsrun** will look for an `inherit` attribute in the `<experiment>` tag. If an `inherit` attribute is found, **rtsmake** and **rtsrun** will look inside the given experiment for the data. It will recurse in this manner until the data is found or until there are no more experiments from which to inherit.
- If the data is not found in the inheritance tree, the value will be empty. If the value was required, an error message will be printed. If the value was optional, a warning message will be printed if you use the **-v** option on **rtsmake** and **rtsrun**.

5.1. Special Case: Namelists

Namelists are parsed and will be given priority as follows:

1.
2.
3.
4.

A warning will be printed if you try to specify a namelist more than once. It is currently not possible to inherit only some values from a given namelist.

5.2. Special Case: Field Tables

Field tables are currently not parsed and are inherited on the basis of their file names. If you specify at least one field table, no field tables will be inherited from the parent experiment.

5.3. Special Case: Executable

The name of the executable has a default value if not specified anywhere. The default location is `$root/$name/exec/fms_$name.x`.

The rts-scripts decide whether a child experiment should have its own executable based on whether you have specified new data in the <cvs> or <compile> sections of the xml for your child experiment. If you intend for your experiment to inherit an executable, you should not re-specify anything in the <cvs> or <compile> sections of your child experiment, because then rtsmake will think you wanted to recompile with the new data. Actually, you do not need to run **rtsmake** on experiments which inherit an executable since the sole purpose of rtsmake is to create an executable.

5.4. Special Case: mkmf template

The location of the mkmf template is a special case because it has a default value if not specified anywhere. The default location is `/home/fms/bin/mkmf.template.$platform`.

6. Tips and Hints

- You may wish to make other directories inside your `$root/$name` experiment directories, such as

```
$root
`-- am2p11
    |-- src   #created by rtsmake for cvs checkouts
    |-- exec  #created by rtsmake for compilation.
    |-- dev   #for code files under development.  List in <srcList> in xml.
    `-- input #for input files like diag_table, etc.
```

- <codeBase> and <modelConfig> are used to construct the first cvs checkout command. The rest of the cvs commands go in <cvsUpdates>, which can handle any csh commands.
- You can use $root and $name in your xml elements. You will eventually also be able to set up other variables, but this has not yet been implemented.
- rtsrun sets the values of $day and $month in coupler_nml.
- om2 users: you can't just list the 'shared' directory currently in the srcList element because this list gets passed directly to mkmf, and mkmf doesn't search directories recursively. The optimal way to handle this is to check out mom4, rename the path_names file, and then check out the shared directory. The resulting path_names file will be used to compile shared code.
- We're working on getting rtscheck to operate with netcdf files. In the meantime Matt has provided an ocean.res file with a checksum for comparison.
- If you want to run several configurations of the same model (using the same executable but different namelists or other input files) you don't need to check out and compile the same code multiple times. Compile once and use the 'inherit' attribute on the remaining experiments.
- For information on the naming of output directories, see Appendix F.

7. Future Work

The RTS is a work in progress, but should be stable and usable. Planned improvements include:

These are done. Let me know if you find bugs.

Please let me know if you have problems or suggestions (arl@gfdl.noaa.gov [mailto:arl@gfdl.noaa.gov]).

A. Usage Information: rtsmake

```
Synopsis: rtsmake checks for the existence of the code directory for each experiment
          (root/experiment_name/src), and if it is not found, executes
          the cvs commands from your xml file.  It then creates a simple compile
          script based on a c-shell template and variables from your xml file.

Usage:    rtsmake [ -s -v -x xmlfile ] experiment [ experiment2 ... ]

          -s         = automatically submit the script with qsub
          -v         = verbose flag
          -x xmlfile = use alternative xml file (default: rts.xml)
          -f         = force rtsmake to run cvs commands, even if src directory exists
          -n         = don't run cvs commands, even if no src directory exists
          -t         = use trap_unititialized and other debugging FFLAGS
          experiment = experiment to create scripts for; must be found in xml file
```

B. Usage Information: rtsrun

```
Synopsis: rtsrun creates a runscript based on a runscript template
          and variables from an xml file.

Usage:    rtsrun [ -s -v -t -x xmlfile -r name ] experiment [ experiment2 ... ]

          -s         = automatically submit the runscripts with qsub -l cpuset
          -v         = verbose flag
          -t         = use the executable created with 'rtsmake -t' which uses
                       trap_unititialized and other debugging FFLAGS
          -x xmlfile = use alternative xml file (default: rts.xml)
          -r name    = perform regression testing using the XML marked 'name'
                       The keyword "suite" will do: basic, restarts, scaling.
                         basic = one 8-day run
                         scaling = one 8-day run on various pe-counts
                         restarts = two 4-day runs and four 2-day runs on the
                                    same number of pes as the 'basic' run
                         production = uses run parameters from database
          experiment = experiment to create scripts for; must be found in xml file
```

C. Usage Information: rtscheck

```
Synopsis: rtscheck runs the resdiff command to compare restart files on the
          output produced by rtsrun-generated runs and prints a report.  It
          also generates a table listing of the runtime for different
          processor counts by parsing the fms.out files.

Usage:    rtscheck [ -x xmlfile ] experiment [ experiment2 ... ]

          -x xmlfile = use alternative xml file (default: rts.xml)
          experiment = experiment to check; must be found in xml file
```

D. Usage Information: rtslist

```
Synopsis: rtslist lists the experiments in your xml file.

Usage:    rtslist [ -v -x xmlfile ]

          -v         = verbose (print experiment descriptions)
          -x xmlfile = use alternative xml file (default: rts.xml)
```

E. Usage Information: rtsstatus

```
Synopsis: rtsstatus parses batch script stdout to relay the status
          of your rtsmake and rtsrun shell scripts.  Note that if you ran
          the scripts interactively, no status information will be found
          since there will be no batch script stdout files.

Usage:    rtsstatus [ -c -r -h -x xmlfile ] experiment [ experiment2 ... ]

          -c         = show compile status only
          -r         = show run status only
          -h         = show this help message, then exit
          -x xmlfile = use alternative xml file (default: rts.xml)
          experiment = experiment to get status of; must be found in xml file
```

F. Naming conventions

1. CVS, Compilation Naming Conventions

The program **rtsmake** creates a CVS checkout script in `$root/scripts/cvs_$name` where `$root` is defined at the top of your `rts.xml` file and `$name` is the experiment name. The CVS checkout script checks out source code into `$root/$name/src`.

The compilation script is then created and placed in `$root/scripts/mk_$name`. The executable created by the script will be `$root/$name/exec/fms_$name.x`.

2. Runscript Naming Conventions

The program **rtsrun** will create one or more runscripts. There are two methods of determining the name for a runscript based on whether you are running regression tests (**rtsrun** is invoked with the **-r regression_name** argument) or a production run (**rtsrun** is invoked without the **-r regression_name** argument).

For production runs, **rtsrun** will create the runscript as `$root/scripts/$name`, deriving the runtime information from the `<production>` element(s) in your XML file. An example `<production>` element is shown here.

```
   <runtime>
      <production simTime="8" units="years" npes="45">
         <segment simTime="1" units="months" runTime="00:44:00"/>
         <ncCombine segments="12" runTime="00:40:00"/>
         <ncAverage segments="12"/>
      </production>
   </runtime>
```

The production runscript will run the full simulation time of 8 years in 1 month segments as denoted above, restarting itself as needed every 8 hours of run time. For explanation of how the production element XML is translated to runtime information in the runscript, see http://www.gfdl.noaa.gov/~arl/rts_example.

For regression tests, **rtsrun** will derive the runtime information from the `<regression>` element(s) in your XML file. Example `<regression>` elements are shown here.

```
   <runtime>
      <regression name="basic">
         <run days="8" npes="15" runTimePerJob="00:30:00"/>
      </regression>
      <regression name="restarts">
         <run days="4 4" npes="15" runTimePerJob="00:20:00"/>
         <run days="2 2 2 2" npes="15" runTimePerJob="00:20:00"/>
      </regression>
      <regression name="scaling">
         <run days="8" npes="1" atmos_layout="1,0" ice_layout="1,0" runTimePerJob="04:00:00"/>
         <run days="8" npes="3" runTimePerJob="02:00:00"/>
         <run days="8" npes="45" runTimePerJob="00:20:00"/>
         <run days="8" npes="60" runTimePerJob="00:20:00"/>
      </regression>
   </runtime>
```

To run a regression test with the information in the regression element labeled "basic" above, use **rtsrun -r basic $name**. Then a runscript will be created at `$root/scripts/$name_$runparams`, where `$runparams` is a string determined by the length of the run, the number of times the executable is called within the script, and the number of processors used. In the "basic" example above, `$runparams` would be `1x0m8d_15pe`, which translates to "one times zero months, eight days on 15 processors".

A single **rtsrun** command may create more than one runscript for a given experiment. The runscripts will have different `$runparams` strings. With the example XML above, **rtsrun -r restarts $name** would create two runscripts, and **rtsrun -r scaling $name** would create four runscripts. The program **rtsrun** also recognizes the keyword `suite`, which would create runscripts from each of the three regression elements `basic, restarts` and `scaling`.

3. Runscript Output Naming Conventions

Output directories are placed in `$archive/$name/` where `$archive` is specified at the top of your `rts.xml` file and `$name` is the experiment name. Production output is placed in three directories directly in `$archive/$name/`. For example, if you specified the following in your `rts.xml` file:

```
<setup>
   <directory type="archive">/archive/fms/rts</directory>
</setup>
```

then production output would be as follows for experiment am2p10:

```
/archive/fms/rts/am2p10/
|-- ascii
|-- history
|-- restart
```

Regression test output utilizes another output directory level under `$archive/$name/` named for the runtime information as described above. The example regression elements shown in the previous section would produce the following output structure:

```
/archive/fms/rts/am2p10/
|-- 1x0m8d_15pe
|   |-- ascii
|   |-- history
|   `-- restart
|-- 1x0m8d_1pe
|   |-- ascii
|   |-- history
|   `-- restart
|-- 1x0m8d_3pe
|   |-- ascii
|   |-- history
|   `-- restart
|-- 1x0m8d_45pe
|   |-- ascii
|   |-- history
|   `-- restart
|-- 1x0m8d_60pe
|   |-- ascii
|   |-- history
|   `-- restart
|-- 1x1m0d_45pe
|   |-- ascii
|   |-- history
|   `-- restart
|-- 2x0m4d_15pe
|   |-- ascii
|   |-- history
|   `-- restart
`-- 4x0m2d_15pe
```

```
|-- ascii
|-- history
`-- restart
```